The Connection of Antipatterns and Maintainability in Firefox

Dénes Bán

The theory that antipatterns have a negative effect on source code maintainability is widely accepted but has relatively little objective research to support it. We aim to extend this research by analyzing the connection of antipatterns and maintainability in an empirical study of Firefox, an open source browser application developed in C++.

First, we selected 45 evenly distributed sample revisions from the `master` and `electrolysis` branches of Firefox between 2009 and 2010 – approximately one revision every two weeks. These provided the basis for both antipattern detection and maintainability assessment. We extracted the occurrences of 3 different antipattern types – Feature Envy, Long Function and Shotgun Surgery – defined by Brown at al. [1], using the same tool we previously published [2]. The antipatterns from the literature were interpreted in this case as violations of specific metric thresholds and/or structural constraints. E.g., for a method to be considered an instance of Feature Envy, it had to have more attribute accesses than a configurable limit while simultaneously the ratio of these accesses referring to foreign – i.e., non-local – attributes had to be higher than another limit. This would lead us to believe that the method itself is interested in attributes, but mainly not the ones belonging to its parent class, hence we call it "envious" of the other classes.

After extracting all the antipatterns, we summed the number of matches by type and divided them by the total number of logical lines of the subject system for each revision to create new, system-level antipattern density predictor metrics. This division makes sure that a system can be considered more maintainable than another even when it has more antipattern instances given that the ratio of these instances to the size of the system is less.

Next, we computed corresponding maintainability values using a C++ specific version of the probabilistic quality model published by Bakota et al. [3], which calculates increasingly more abstract source code characteristics by performing a weighted aggregation of lower level metrics according to the ISO/IEC 25010 standard [4]. Its final result is a number between 0 and 1 indicating the maintainability of the source code.

After this, we checked for correlations between the maintainability values and each of the different antipattern densities to unveil a connection between the underlying concepts. We found that Feature Envy, Long Function and Shotgun Surgery had coefficients of -0.95, -0.94 and -0.36 for Pearson's, and -0.83, -0.85 and -0.57 for Spearman's correlation, respectively. These figures show strong, inverse relationships, thereby supporting our initial assumption that the more antipatterns the source code contains, the harder it is to maintain.

Finally, we combined these data into a table applicable for machine learning experiments, which we conducted using Weka [5] and a number of its built-in algorithms. All regression types we tried for predicting source code maintainability from antipattern information – namely, Linear Regression, Multilayer Perceptron, REPTree, M5P and SMO Regression – reached correlation coefficients over .93 while using negative weights for the antipattern predictors in their built models. This not only means we can give precise estimations for the maintainability of the source code using its antipattern densities alone but also emphasizes their reversed roles.

In conclusion, we believe that this empirical study is another step towards objectively proving that antipatterns have an adverse effect on software maintainability.

References

[1] Brown, W.J., Malveau, R.C., McCormick III, I.H.W., Mowbray, T.J.: AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis. *John Wiley & Sons, Inc.*, 1998.

[2] Dénes Bán, Rudolf Ferenc: Recognizing Antipatterns and Analyzing their Effects on Software Maintainability, *In Proceedings of the 14th International Conference on Computational Science and Its Applications*, 2014.

[3] Tibor Bakota, Péter Hegedűs, Péter Körtvélyesi, Rudolf Ferenc, and Tibor Gyimóthy: A Probabilistic Software Quality Model. *In Proceedings of the 27th IEEE International Conference on Software Maintenance*, 2011.

[4] ISO/IEC: ISO/IEC 25000:2005. Software Engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE. 2005.

[5] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten: The WEKA Data Mining Software: An Update. `SIGKDD Explorations`, 2009.